Extended precision on the CELL processor

Diep Nguyen Hong  Stef Graillat  Jean-Luc Lamotte

LIP6/PEQUAN
P. and M. Curie University

REC’08, Third International Worshhop on Reliable Engineering Computing
Savannah, Georgia, USA, February 20-22, 2008
1. The Cell processor
2. Reliable computing and extended precision on Cell processor
3. Results
4. Conclusions
The CELL processor

SP $> 200 \text{ GFlops}$, DP$=15 \text{ Gflops}$, $25\text{GB/s}$ memory BW, $300 \text{ GB/s}$ EIB
Power Processor Element (PPE)

The PPE is based on the 2-way Power Architecture with:
- 32 KB of L1 cache for instructions
- 32 KB of L1 cache for data
- 512 KB of L2 cache

The PPE is fully pipelined for double precision computation and fully IEEE compliant.
The SPE is a small processor with a vectorial unit.

- small memory (256 KB) for instructions and data, named “local store” (LS)
- 128 registers of 128 bits
- 1 SPU “Synergistic Processing Unit”
  - 4 units for single precision computation
  - 1 unit for double precision computation
- MFC “Memory Flow Controller” which manages memory access through DMA
128-bit registers:
- 16 integers of 8-bits,
- 8 integers of 16-bits,
- 4 integers of 32-bits,
- 4 single precision floating point numbers,
- 2 double precision floating point numbers.

The SIMD processor is based on FMA and is fully pipelined in SP:

Peak performance SP: \(4 \times 2 \times 3.2 = 25.6\, GFLOPs\)

Not fully pipelined in double precision:

Peak performance in DP: \(2 \times 2 \times 3.2/7 = 1.8\, GFLOPs\)
Parallelism on CELL

3 levels of parallelism:

1. processes on CELL processors, exchange with a MPI library,
2. threads on 8 SPE,
3. inside a thread, SIMD programming.
3 levels of parallelism:

1. processes on CELL processors, exchange with a MPI library,
2. threads on 8 SPE,
3. inside a thread, SIMD programming.
Parallelism on CELL

3 levels of parallelism:
1. processes on CELL processors, exchange with a MPI library,
2. threads on 8 SPE,
3. inside a thread, SIMD programming.
Data distribution and communication between PPE and SPE:

- ALF
- mailing box
- exchange through DMA
- data need to be aligned on quadword
- double buffering technique

on an SPE

- only 256 KB
- Altivec programming
- code and data dependencies: not to break the SIMD pipeline
No division

$1/x$ and $1/\sqrt{x}$: only the 12 first bits are exact.

SPU float arithmetic is not IEEE compliant:

- only rounding mode to zero (truncation).
- The highest exponent (128) is used not for Infinity or NaN, but to extend the range of the floating point.
- Inf and NaN are not recognized by arithmetic operations.
- Overflow results saturate to the largest representable positive or negative values, rather than producing $+/-$IEEE Infinity.
- No denormalized results: $+0$ instead.
SPU double arithmetic is IEEE compliant except:

- FP trapping is not supported.
- Denormalized operands are treated as 0.
- NaN results are always the default QNaN (Quiet NaN)
Reliable computing on Cell processor

difficult to implement interval arithmetic.

possible to “emulate” a rounding mode toward $+\infty$
if $r \in \mathbb{R}$ non-negative, $\text{fl}_0(r) \leq r \leq \text{succ}(\text{fl}_0(r))$
and

$$\text{succ}(f) = \max\{\text{fl}_0(f + 2uf), \text{fl}_0(f + u)\}.$$ 

where $u$ is the relative rounding error and $u$ the underflow unit.
Error-free transformations

Let $a, b \in \mathbb{F}$, and $\circ$ an operation in $\circ \in \{+,-,\cdot, /\}$

\[(a \circ b) \in \mathbb{R} \quad \not\in \mathbb{F}\]

\[\rightarrow \text{fl}(a \circ b) \neq (a \circ b)\]

$(a \circ b) - \text{fl}(a \circ b) = \text{err}$ is the roundoff error

“Error-free transformation” (EFT) : allows us to find the couple $(x, y)$ such as :

- $x \approx \text{fl}(a \circ b)$
- $a \circ b = x + y$
EFT for the sum with rounding mode to nearest

\[ x = \text{fl}(a \pm b) \implies a \pm b = x + y \text{ with } y \in \mathbb{F}, \]

Algorithm 1 (EFT for the sum of 2 floating point numbers (Knuth 1969))

\[
\text{function } [x, y] = \text{TwoSum}(a, b) \\
\quad x = \text{fl}(a + b) \\
\quad z = \text{fl}(x - a) \\
\quad y = \text{fl}((a - (x - z)) + (b - z))
\]

Cost : 6 FLOPs

Algorithm 2 (EFT for the sum of 2 floating point numbers (Dekker 1971), \(|a| \geq |b|\))

\[
\text{function } [x, y] = \text{FastTwoSum}(a, b) \\
\quad x = \text{fl}(a + b) \\
\quad y = \text{fl}((a - x) + b)
\]

Cost : 3 FLOPs
Algorithm 3 (EFT for the sum of 2 floating point numbers with a rounding mode toward zero (Priest))

function \([x, y] = \text{TwoSum} \rightarrow \text{toward} \rightarrow \text{zero}(a, b)\)

if \(|b| > |a|\)

\[\text{swap}(a, b)\]

\[x = \text{fl}(a + b)\]

\[d = \text{fl}(x - a)\]

\[y = \text{fl}(b - d)\]

if \(y + d \neq b\)

\[x = a, y = b\]

Cost : 6.5 FLOPs
EFT for the sum with rounding mode toward zero

Algorithm 4 (EFT for the sum of 2 floating point numbers with a rounding mode toward zero)

function \([x, y] = \text{TwoSum} - \text{toward} - \text{zero}(a, b)\)

\[
\begin{align*}
\text{if } (|b| > |a|) \\
\quad &\text{swap}(a, b) \\
\quad x = \text{fl}(a + b) \\
\quad d = \text{fl}(x - a) \\
\quad y = \text{fl}(b - d) \\
\text{if } (|2\times b| < |d|) \\
\quad x = a, y = b
\end{align*}
\]

Cost : 6.5 FLOPs

Theorem 1

*The algorithm TwoSum – toward – zero transforms 2 floating point numbers \(a\) and \(b\) into a couple of floating point numbers \((x, y)\) satisfying*

\[x + y = a + b \text{ and } |y| < \text{ulp}(x)\]
EFT for the product with rounding mode to nearest

\[ x = \text{fl}(a \cdot b) \implies a \cdot b = x + y \quad \text{with } y \in \mathbb{F}, \]

Algorithm TwoProduct of Veltkamp and Dekker (1971)

\[ a = x + y \quad \text{and} \quad x \text{ and } y \text{ non-overlapping with } |y| \leq |x|. \]

Algorithm 5 (Error-free split of a floating point number into two parts)

```plaintext
function \([x, y] = \text{Split}(a)\) 
\quad \text{factor} = \text{fl}(2^s + 1) \mod \text{u} = 2^{-p}, \; s = \lceil p/2 \rceil 
\quad c = \text{fl}(\text{factor} \cdot a) 
\quad x = \text{fl}(c - (c - a)) 
\quad y = \text{fl}(a - x) 
```

Cost: 4 FLOPs
Algorithm 6 (EFT of the product of two floating point numbers)

function $[x, y] = \text{TwoProduct}(a, b)$

$x = \text{fl}(a \cdot b)$

$[a_1, a_2] = \text{Split}(a)$

$[b_1, b_2] = \text{Split}(b)$

$y = \text{fl}(a_2 \cdot b_2 - (((x - a_1 \cdot b_1) - a_2 \cdot b_1) - a_1 \cdot b_2))$

Cost : 17 FLOPs
EFT for the product with rounding mode to nearest

What is a Fused Multiply and Add (FMA) in floating point arithmetic?

→ Given \( a, b \) and \( c \), three floating point numbers, \( \text{FMA}(a, b, c) \) computes \( a \cdot b + c \) rounded according to the current rounding mode

⇒ only one rounding error for two operations!

FMA is available Cell processors.

Algorithm 7 (EFT of the product of two floating point numbers)

function \([x, y] = \text{TwoProductFMA}(a, b)\)

\[
x = \text{fl}(a \cdot b)
\]

\[
y = \text{FMA}(a, b, -x)
\]

⇒ Still valid with rounding toward zero!

Cost : 2 FLOPs
Definition 1 (extended precision)

An extended precision number of n is a non-evaluated sum of n floating point number. \( x = x_1 + x_2 + \ldots + x_n \)

Normalisation:

1. to the nearest: \( |x_{k+1}| \leq \frac{1}{2} ulp(x_k) \).
2. toward zero: \( |x_{k+1}| < ulp(x_k) \) have the same sign.

Precision used on Cell processor: simple precision

- \( n=2 \): double-simple
Theorem 2

Let $a = a_h + a_l$ and $b = b_h + b_l$, two double-simples to add, $r = r_h + r_l$ the result and $\delta$ the algorithm error. The algorithm error satisfies

$$r = a + b + \delta$$

$$|\delta| < \max(2^{-23} \cdot |a_l + b_l|, 2^{-43} \cdot |a_h + a_l + b_h + b_l|) + 2^{-45} \cdot |a + b|.$$
The exact transformation code

\[ a, b : \text{vector of 4 floating point numbers.} \]

1. \textbf{TwoSum-toward-zero} \((a,b)\)
2. \(\text{comp} = \text{spu\_cmpabsgt}(b,a)\)
3. \(\text{hi} = \text{spu\_sel}(a, b, \text{comp})\)
4. \(\text{lo} = \text{spu\_sel}(b, a, \text{comp})\)
5. \(s = \text{spu\_add}(a, b)\)
6. \(d = \text{spu\_sub}(s, \text{hi})\)
7. \(e = \text{spu\_sub}(\text{lo}, d)\)
8. \(\text{tmp} = \text{spu\_mul}(2, \text{lo})\)
9. \(\text{comp} = \text{spu\_cmpabsgt}(d, \text{tmp})\)
10. \(s = \text{spu\_sel}(s, \text{hi}, \text{comp})\)
11. \(e = \text{spu\_sel}(e, \text{lo}, \text{comp})\)
12. \text{return } (s,e)\]

\textbf{Cost : 20 cycles}
Renormalisation

1  | Renormalise2-toward-zero (a,b)
2   | s = spu_add(a, b)
3   | comp = spu_cmpabsgt(b,a)
4   | hi = spu_sel(a, b, comp)
5   | lo = spu_sel(b, a, comp)
6   | d = spu_sub(s, hi)
7   | e = spu_sub(lo, d)
8   | return (s,e)

Cost : 18 cycles
Theorem 3

Let $a$ and $b$ be two single floating point numbers. The result returns by $\text{Renormalise2-toward-zero}$ is a double simple number $(s, e)$ which satisfies

- $s$ and $e$ have the same sign
- $|e| < ulp(s)$
- $a + b = s + e + \delta$ with $\delta \leq 2^{-45}|a + b|$. 
Addition of two double-simple: the natural version

\begin{align*}
\text{add_ds_ds_vect}(a, b) & = (s, e) = \text{TwoSum-toward-zero}(a, b) \\
& \quad t = \text{spu_shuffle}(s, s, \text{switch-vect}) \\
& \quad t_1 = \text{spu_add}(t, e) \\
& \quad (hi, lo) = \text{Renormalise2-toward-zero}(s, t_1) \\
& \quad \text{res} = \text{spu_shuffle}(hi, lo, \text{merge-vect}) \\
& \quad \text{return res}
\end{align*}

Cost: 50 cycles / 2 operations
add ds ds 2vect (vect_a1, vect_a2, vect_b1, vect_b2)

a_hi = spu_shuffle(vect_a1, vect_a2, _merge1_vect_)
a_lo = spu_shuffle(vect_a1, vect_a2, _merge2_vect_)
b_hi = spu_shuffle(vect_b1, vect_b2, _merge1_vect_)
b_lo = spu_shuffle(vect_b1, vect_b2, _merge2_vect_)

(s, e) = TwoSum-toward-zero (a_hi, b_hi)
t1 = spu_add(a_lo , b_lo)
tmp = spu_add(t1 , e)

(hi, lo) = Renormalise2-toward-zero (s , tmp)
vect_c1 = spu_shuffle(hi, lo, _merge1_vect_)
vect_c2 = spu_shuffle(hi, lo, _merge2_vect_)

return (vect_c1, vect_c2)

Cost : 64 cycles / 4 opérations
The version 2 increases the performance of the sum. Cycles are still lost.

⇒ to perform version 2 twice in a same function.

Cost: 72 cycles / 8 operations
Theoretical results

frequency: 3.2GHz.
The peak performance in double precision: \(2 \times 2 \times 3.2/7 = 1.8\text{GFLOPs} \).
Comparison with double precision

Addition of two vectors with DMA to load data on SPE

<table>
<thead>
<tr>
<th>MFLOPs</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE double</td>
<td>336</td>
<td>559</td>
<td>932</td>
<td>1089</td>
</tr>
<tr>
<td>double-simple</td>
<td>336</td>
<td>559</td>
<td>932</td>
<td>1089</td>
</tr>
<tr>
<td>Functions</td>
<td>Theoretical (1 SPE)</td>
<td>Measured (1 SPE)</td>
<td>Measured (8 SPEs)</td>
<td></td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------------------</td>
<td>------------------</td>
<td>-------------------</td>
<td></td>
</tr>
<tr>
<td><code>Add_ds_ds_4vect</code></td>
<td>355</td>
<td>266</td>
<td>2133</td>
<td></td>
</tr>
<tr>
<td><code>Mul_ds_ds_4vect</code></td>
<td>406</td>
<td>320</td>
<td>2560</td>
<td></td>
</tr>
<tr>
<td>double precision addition</td>
<td>914</td>
<td>914</td>
<td>7314</td>
<td></td>
</tr>
<tr>
<td>double precision product</td>
<td>914</td>
<td>914</td>
<td>7314</td>
<td></td>
</tr>
</tbody>
</table>

**Tab.**: Performance without data exchange (MFLOPS)
Quad simple

<table>
<thead>
<tr>
<th>function</th>
<th>Cycles number</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add_qs_qs_4vect</td>
<td>449 cycles / 4</td>
<td>28.5 MFLOPs</td>
</tr>
<tr>
<td>Mul_qs_qs_4vect</td>
<td>583 cycles / 4</td>
<td>21.9 MFLOPs</td>
</tr>
</tbody>
</table>
The true goal:
- to prepare the work for the next CELL:
  - fully pipelined double precision floating point number
  - probably 512 KB on SPE
- a double double library,
- a quad double library.
Rumours on the next generation

- IEEE compliant
- from 8 to 32 SPE
- over 1TFLOPS
Yozo Hida, Xiaoye S.Li, and David H.Baily. 
Quad-double arithmetic : Algorithms, implementations, and application. 
2000.

Donald Knuth. 
*The art of computer programming : Seminumerical algorithms* 

Takeshi Ogita, Siegfried M. Rump, and Shin’ichi Oishi. 
Accurate sum and dot product. 

Christoph Quirin Lauter. 
Basic building blocks for a triple-double intermediate format 
Tech. report, INRIA, 2005
Douglas M. Priest.

T.J Dekker.